A Brief Understanding of README.md

A Brief Understanding of README.md

Import Libraries

Code Block

Remarks

Load The Dataset

Code Block

Remarks

Preparing Data

Code Block

Remarks

Load 2D Detections

Code Block

Remarks

Code Block

Remarks

Function Fetch()

Code Block

Remarks

Code Block

Remarks

Code Block

Remarks

Code Block

Remarks

Evaluate (本部分不要求理解,只要会用就行)

Code Block

Remarks

Import Libraries

```
import numpy as np

from common.arguments import parse_args
import torch

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
import os
    import sys
11
    import errno
12
13
   from common.camera import *
14
   from common.model import *
15
   from common.loss import *
   from common.generators import ChunkedGenerator, UnchunkedGenerator
16
17
    from time import time
18
    from common.utils import deterministic_random
```

- common.camera 等引入都是在文件夹 common 下的.py 文件, 例如 common.camera == common/camera.py。
- 对与torch包中三个函数的名字进行了重载,例如使用 F 来代替 torch.nn.functional 来节省空间。

Load The Dataset

```
args = parse args()
 2
    print(args)
 3
    try:
        # Create checkpoint directory if it does not exist
 5
 6
        os.makedirs(args.checkpoint)
 7
    except OSError as e:
 8
        if e.errno != errno.EEXIST:
            raise RuntimeError('Unable to create checkpoint directory:',
    args.checkpoint)
10
11
    print('Loading dataset...')
    dataset_path = 'data/data_3d_' + args.dataset + '.npz'
12
    if args.dataset == 'h36m':
13
        from common.h36m dataset import Human36mDataset
14
15
        dataset = Human36mDataset(dataset path)
    elif args.dataset.startswith('humaneva'):
16
17
        from common.humaneva dataset import HumanEvaDataset
        dataset = HumanEvaDataset(dataset_path)
18
    elif args.dataset.startswith('custom'):
19
20
        from common.custom_dataset import CustomDataset
```

```
dataset = CustomDataset('data/data_2d_' + args.dataset + '_' +
args.keypoints + '.npz')

else:
    raise KeyError('Invalid dataset')
```

- Keyerror 在使用映射中不存在的键时引发
- 先尝试新建 ·/checkpoint 文件夹。如果文件夹已经存在,不再新建;如果还不存在,则新建文件 夹。如果不在这两种情况之内,则报错,终止程序。
- 定义变量 dataset path 为数据集路径变量。观察 data/data 3d 中我们的数据集:
- 根据 README.md 说明, python run.py -e 80 -k cpn_ft_h36m_dbb -arc 3,3,3,3,3是成功 运行的解,这说明我们并没有人为输入 path 参数,而是在 run.py 中已经定义了这个参数。
- 根据 DATASET.md 说明,目录结构已经符合如下样式:

```
/path/to/dataset/S1/MyPoseFeatures/D3_Positions/Directions 1.cdf
/path/to/dataset/S1/MyPoseFeatures/D3_Positions/Directions.cdf
...
```

• 再根据 DATASET.md 指出的 pyscript 脚本 (新版本, 不使用 matlab)

```
cd data
python prepare_data_h36m.py --from-source-cdf /path/to/dataset
cd ..
```

我们看进 prepare_data_h36m.py 的代码:

```
1 import argparse
 2 import os
 3 import zipfile
 4 import numpy as np
 5 import h5py
  from glob import glob
   from shutil import rmtree
 7
   limport sys # 这是system扩展包,用于进行系统的io处理
10 sys.path.append('../')
   from common.h36m dataset import Human36mDataset # 引入../common中写好的
11
    h36m dataset.py脚本
12 from common.camera import world_to_camera, project_to_2d,
    image_coordinates
1.3
   from common.utils import wrap
14
15 | output_filename = 'data_3d_h36m'
   output_filename_2d = 'data_2d_h36m_gt'
16
   subjects = ['S1', 'S5', 'S6', 'S7', 'S8', 'S9', 'S11']
17
18
```

```
19
    if name == ' main ': # execute only if run as a script
20
        if os.path.basename(os.getcwd()) != 'data':
            print('This script must be launched from the "data" directory') #
21
    必须在给定的文件夹启动本脚本
22
            exit(0)
23
24
        parser = argparse.ArgumentParser(description='Human3.6M dataset
    downloader/converter')
25
26
        # Convert dataset preprocessed by Martinez et al. in
    https://github.com/una-dinosauria/3d-pose-baseline
        parser.add_argument('--from-archive', default='', type=str,
2.7
    metavar='PATH', help='convert preprocessed dataset')
28
2.9
        # Convert dataset from original source, using files converted to .mat
    (the Human3.6M dataset path must be specified manually)
30
        # This option requires MATLAB to convert files using the provided
31
        parser.add_argument('--from-source', default='', type=str,
    metavar='PATH', help='convert original dataset')
32
        # Convert dataset from original source, using original .cdf files
33
    (the Human3.6M dataset path must be specified manually)
        # This option does not require MATLAB, but the Python library cdflib
34
    must be installed
        parser.add argument('--from-source-cdf', default='', type=str,
35
    metavar='PATH', help='convert original dataset')
36
37
        args = parser.parse args()
38
39
        if args.from_archive and args.from_source:
40
            print('Please specify only one argument')
41
            exit(0)
42
        if os.path.exists(output filename + '.npz'):
43
            print('The dataset already exists at', output_filename + '.npz')
44
45
            exit(0)
46
        if args.from archive:
47
48
            print('Extracting Human3.6M dataset from', args.from_archive)
            with zipfile.ZipFile(args.from_archive, 'r') as archive:
49
                archive.extractall()
50
51
52
            print('Converting...')
53
            output = {}
            for subject in subjects:
54
55
                output[subject] = {}
                file_list = glob('h36m/' + subject +
56
    '/MyPoses/3D positions/*.h5')
```

```
57
                assert len(file list) == 30, "Expected 30 files for subject "
    + subject + ", got " + str(len(file_list))
58
                for f in file list:
59
                    action = os.path.splitext(os.path.basename(f))[0]
60
                    if subject == 'S11' and action == 'Directions':
61
62
                         continue # Discard corrupted video
63
                    with h5py.File(f) as hf:
64
65
                        positions = hf['3D_positions'].value.reshape(32, 3,
    -1).transpose(2, 0, 1)
                        positions /= 1000 # Meters instead of millimeters
66
67
                         output[subject][action] = positions.astype('float32')
68
            print('Saving...')
69
70
            np.savez compressed(output filename, positions 3d=output)
71
72
            print('Cleaning up...')
73
            rmtree('h36m')
74
75
            print('Done.')
76
77
        elif args.from_source:
78
            print('Converting original Human3.6M dataset from',
    args.from source)
79
            output = {}
80
81
            from scipy.io import loadmat
82
83
            for subject in subjects:
84
                output[subject] = {}
85
                file list = glob(args.from source + '/' + subject +
    '/MyPoseFeatures/D3_Positions/*.cdf.mat')
                assert len(file list) == 30, "Expected 30 files for subject "
86
    + subject + ", got " + str(len(file list))
87
                for f in file list:
88
                    action =
    os.path.splitext(os.path.splitext(os.path.basename(f))[0])[0]
89
                    if subject == 'S11' and action == 'Directions':
90
91
                         continue # Discard corrupted video
92
                    # Use consistent naming convention
93
                    canonical name = action.replace('TakingPhoto', 'Photo') \
94
                                            .replace('WalkingDog', 'WalkDog')
95
96
97
                    hf = loadmat(f)
98
                    positions = hf['data'][0, 0].reshape(-1, 32, 3)
99
                    positions /= 1000 # Meters instead of millimeters
```

```
100
                      output[subject][canonical name] =
     positions.astype('float32')
101
102
             print('Saving...')
103
             np.savez_compressed(output_filename, positions_3d=output)
104
105
             print('Done.')
106
         elif args.from source cdf:
107
108
             print('Converting original Human3.6M dataset from',
     args.from source cdf, '(CDF files)')
109
             output = {}
110
             import cdflib
111
112
113
             for subject in subjects:
114
                 output[subject] = {}
115
                 file list = glob(args.from source cdf + '/' + subject +
     '/MyPoseFeatures/D3 Positions/*.cdf')
                 assert len(file_list) == 30, "Expected 30 files for subject "
116
     + subject + ", got " + str(len(file list))
                 for f in file list:
117
                      action = os.path.splitext(os.path.basename(f))[0]
118
119
                      if subject == 'S11' and action == 'Directions':
120
121
                          continue # Discard corrupted video
122
123
                      # Use consistent naming convention
                      canonical name = action.replace('TakingPhoto', 'Photo') \
124
125
                                              .replace('WalkingDog', 'WalkDog')
126
127
                      hf = cdflib.CDF(f)
128
                      positions = hf['Pose'].reshape(-1, 32, 3)
129
                      positions /= 1000 # Meters instead of millimeters
130
                      output[subject][canonical name] =
     positions.astype('float32')
131
132
             print('Saving...')
             np.savez compressed(output filename, positions 3d=output)
133
134
             print('Done.')
135
136
137
138
             print('Please specify the dataset source')
139
             exit(0)
140
         # Create 2D pose file
141
142
         print('')
143
         print('Computing ground-truth 2D poses...')
```

```
144
         dataset = Human36mDataset(output filename + '.npz') # 使用npz文件存储计
     算出的2d pose gt文件
145
         output 2d poses = {}
         for subject in dataset.subjects():
146
147
             output 2d poses[subject] = {}
148
             for action in dataset[subject].keys():
149
                 anim = dataset[subject][action]
150
151
                 positions 2d = []
152
                 for cam in anim['cameras']:
153
                     pos 3d = world to camera(anim['positions'],
     R=cam['orientation'], t=cam['translation'])
                     pos_2d = wrap(project_to_2d, pos_3d, cam['intrinsic'],
154
     unsqueeze=True)
                     pos_2d_pixel_space = image_coordinates(pos_2d,
155
     w=cam['res_w'], h=cam['res_h'])
156
                     positions_2d.append(pos_2d_pixel_space.astype('float32'))
157
                 output 2d poses[subject][action] = positions 2d
158
159
         print('Saving...')
160
         metadata = {
             'num joints': dataset.skeleton().num joints(),
161
              'keypoints_symmetry': [dataset.skeleton().joints_left(),
     dataset.skeleton().joints_right()]
163
         np.savez compressed(output filename 2d, positions 2d=output 2d poses,
164
     metadata=metadata)
165
166
         print('Done.')
```

发现最后所有的data都被装进了一个文件 data/data_3d_h36m.npz 中。我们跟着看 README.md 中剩下来的代码:

```
dataset_path = 'data/data_3d_' + args.dataset + '.npz'
    if args.dataset == 'h36m':
 2
 3
        from common.h36m dataset import Human36mDataset
 4
        dataset = Human36mDataset(dataset path)
    elif args.dataset.startswith('humaneva'):
 5
 6
        from common.humaneva dataset import HumanEvaDataset
 7
        dataset = HumanEvaDataset(dataset path)
8
    elif args.dataset.startswith('custom'):
        from common.custom dataset import CustomDataset
 9
        dataset = CustomDataset('data/data 2d ' + args.dataset + ' ' +
10
    args.keypoints + '.npz')
    else:
11
12
        raise KeyError('Invalid dataset')
```

● 发现程序已经选择了 Human3.6mDataset(dataset path) 进行挂载。

Preparing Data

Code Block

```
print('Preparing data...')
 2
    for subject in dataset.subjects():
 3
        for action in dataset[subject].keys():
            anim = dataset[subject][action]
 4
            if 'positions' in anim:
                positions 3d = []
                for cam in anim['cameras']:
 8
 9
                    pos_3d = world_to_camera(anim['positions'],
    R=cam['orientation'], t=cam['translation'])
10
                    pos_3d[:, 1:] -= pos_3d[:, :1] # Remove global offset, but
    keep trajectory in first position
11
                    positions_3d.append(pos_3d)
12
                anim['positions 3d'] = positions 3d
```

Remarks

- 对于数据集中每一个对象都执行准备(预处理)操作。
- 删除全局偏移,但将轨迹保持在第一位置。

Load 2D Detections

```
print('Loading 2D detections...')
   keypoints = np.load('data/data_2d_' + args.dataset + '_' + args.keypoints
    + '.npz', allow_pickle=True)
    keypoints metadata = keypoints['metadata'].item()
    keypoints_symmetry = keypoints_metadata['keypoints_symmetry']
    kps_left, kps_right = list(keypoints_symmetry[0]),
    list(keypoints_symmetry[1])
    joints left, joints right = list(dataset.skeleton().joints left()),
    list(dataset.skeleton().joints right())
7
    keypoints = keypoints['positions_2d'].item()
8
    for subject in dataset.subjects():
        assert subject in keypoints, 'Subject {} is missing from the 2D
10
    detections dataset'.format(subject)
        for action in dataset[subject].keys():
11
            assert action in keypoints[subject], 'Action {} of subject {} is
12
    missing from the 2D detections dataset'.format(action, subject)
            if 'positions_3d' not in dataset[subject][action]:
13
```

```
14
                continue
15
            for cam_idx in range(len(keypoints[subject][action])):
16
17
18
                # We check for >= instead of == because some videos in H3.6M
    contain extra frames
19
                mocap_length = dataset[subject][action]['positions_3d']
    [cam idx].shape[0]
20
                assert keypoints[subject][action][cam_idx].shape[0] >=
    mocap_length
21
                if keypoints[subject][action][cam_idx].shape[0] >
2.2
    mocap_length:
23
                    # Shorten sequence
24
                    keypoints[subject][action][cam_idx] = keypoints[subject]
    [action][cam_idx][:mocap_length]
25
26
            assert len(keypoints[subject][action]) == len(dataset[subject]
    [action]['positions 3d'])
```

• 装载2D探测。根据命令 keypoints = np.load('data/data_2d_' + args.dataset + '_' + args.keypoints + '.npz', allow_pickle=True) 推知



主要的变化是 args.keypoints, 因为custom只在inference in the wild的时候才会用到。对于

```
for subject in keypoints.keys():
 2
        for action in keypoints[subject]:
 3
            for cam idx, kps in enumerate(keypoints[subject][action]):
                # Normalize camera frame
 4
 5
                cam = dataset.cameras()[subject][cam_idx]
                kps[..., :2] = normalize_screen_coordinates(kps[..., :2],
 6
    w=cam['res_w'], h=cam['res_h'])
 7
                keypoints[subject][action][cam idx] = kps
 8
9
    subjects_train = args.subjects_train.split(',')
    subjects semi = [] if not args.subjects unlabeled else
10
    args.subjects_unlabeled.split(',')
11
    if not args.render:
        subjects test = args.subjects test.split(',')
```

```
13 else:
14     subjects_test = [args.viz_subject]
15
16     semi_supervised = len(subjects_semi) > 0
17     if semi_supervised and not dataset.supports_semi_supervised():
18         raise RuntimeError('Semi-supervised training is not implemented for this dataset')
```

• RuntimeError



Function Fetch()

```
def fetch(subjects, action filter=None, subset=1, parse 3d poses=True):
 1
 2
        out poses 3d = []
 3
        out_poses_2d = []
 4
        out_camera_params = []
 5
        for subject in subjects:
            for action in keypoints[subject].keys():
                 if action filter is not None:
                     found = False
 8
9
                     for a in action filter:
10
                         if action.startswith(a):
11
                             found = True
                             break
12
                     if not found:
13
14
                         continue
15
                 poses 2d = keypoints[subject][action]
16
                 for i in range(len(poses_2d)): # Iterate across cameras
17
                     out_poses_2d.append(poses_2d[i])
18
19
2.0
                 if subject in dataset.cameras():
21
                     cams = dataset.cameras()[subject]
22
                     assert len(cams) == len(poses_2d), 'Camera count mismatch'
23
                     for cam in cams:
                         if 'intrinsic' in cam:
25
                             out_camera_params.append(cam['intrinsic'])
26
2.7
                 if parse_3d_poses and 'positions_3d' in dataset[subject]
    [action]:
```

```
28
                     poses 3d = dataset[subject][action]['positions 3d']
29
                     assert len(poses_3d) == len(poses_2d), 'Camera count
    mismatch'
                    for i in range(len(poses 3d)): # Iterate across cameras
30
31
                         out_poses_3d.append(poses_3d[i])
32
33
        if len(out_camera_params) == 0:
            out camera params = None
34
35
        if len(out_poses_3d) == 0:
36
            out_poses_3d = None
37
38
        stride = args.downsample
        if subset < 1:
39
40
            for i in range(len(out_poses_2d)):
41
                n_frames = int(round(len(out_poses_2d[i])//stride *
    subset)*stride)
42
                start = deterministic_random(0, len(out_poses_2d[i]) -
    n frames + 1, str(len(out poses 2d[i])))
43
                out_poses_2d[i] = out_poses_2d[i][start:start+n_frames:stride]
                if out_poses_3d is not None:
44
45
                    out poses 3d[i] = out poses 3d[i]
    [start:start+n frames:stride]
        elif stride > 1:
47
            # Downsample as requested
48
            for i in range(len(out poses 2d)):
                out_poses_2d[i] = out_poses_2d[i][::stride]
49
50
                if out_poses_3d is not None:
51
                    out_poses_3d[i] = out_poses_3d[i][::stride]
52
        return out_camera_params, out_poses_3d, out_poses_2d
```

```
action_filter = None if args.actions == '*' else args.actions.split(',')
if action_filter is not None:
    print('Selected actions:', action_filter)

cameras_valid, poses_valid, poses_valid_2d = fetch(subjects_test, action_filter)
```

```
filter widths = [int(x) for x in args.architecture.split(',')]
    if not args.disable optimizations and not args.dense and args.stride == 1:
9
        # Use optimized model for single-frame predictions
1.0
        model pos train =
    TemporalModelOptimized1f(poses valid 2d[0].shape[-2],
    poses_valid_2d[0].shape[-1], dataset.skeleton().num_joints(),
11
                                    filter widths=filter widths,
    causal=args.causal, dropout=args.dropout, channels=args.channels)
12
    else:
13
        # When incompatible settings are detected (stride > 1, dense filters,
    or disabled optimization) fall back to normal model
        model_pos_train = TemporalModel(poses_valid_2d[0].shape[-2],
14
    poses_valid_2d[0].shape[-1],
    dataset.skeleton().num_joints(),filter_widths=filter_widths,causal=args.ca
    usal,dropout=args.dropout,channels=args.channels,dense=args.dense)
```

```
model_pos = TemporalModel(poses_valid_2d[0].shape[-2],
    poses_valid_2d[0].shape[-1], dataset.skeleton().num_joints(),
                                 filter widths=filter widths,
    causal=args.causal, dropout=args.dropout, channels=args.channels,
 3
                                 dense=args.dense)
 5
    receptive field = model pos.receptive field()
    print('INFO: Receptive field: {} frames'.format(receptive field))
    pad = (receptive_field - 1) // 2 # Padding on each side
 8
    if args.causal:
9
        print('INFO: Using causal convolutions')
10
        causal shift = pad
11
12
        causal_shift = 0
13
    model params = 0
14
    for parameter in model pos.parameters():
15
16
        model params += parameter.numel()
17
    print('INFO: Trainable parameter count:', model_params)
18
19
    if torch.cuda.is_available():
20
        model_pos = model_pos.cuda()
2.1
        model_pos_train = model_pos_train.cuda()
```

```
22
    if args.resume or args.evaluate:
23
        chk filename = os.path.join(args.checkpoint, args.resume if
24
    args.resume else args.evaluate)
2.5
        print('Loading checkpoint', chk_filename)
        checkpoint = torch.load(chk filename, map location=lambda storage,
26
    loc: storage)
        print('This model was trained for {}
27
    epochs'.format(checkpoint['epoch']))
28
        model_pos_train.load_state_dict(checkpoint['model_pos'])
29
        model pos.load state dict(checkpoint['model pos'])
30
        if args.evaluate and 'model_traj' in checkpoint:
31
            # Load trajectory model if it contained in the checkpoint (e.g.
32
    for inference in the wild)
            model traj = TemporalModel(poses_valid_2d[0].shape[-2],
33
    poses_valid_2d[0].shape[-1], 1,
34
                                 filter widths=filter widths,
    causal=args.causal, dropout=args.dropout, channels=args.channels,
35
                                 dense=args.dense)
36
            if torch.cuda.is available():
                model traj = model traj.cuda()
37
            model_traj.load_state_dict(checkpoint['model_traj'])
38
39
        else:
40
            model traj = None
```

```
test generator = UnchunkedGenerator(cameras valid, poses valid,
    poses_valid_2d,
                                         pad=pad, causal_shift=causal_shift,
    augment=False,
3
                                         kps_left=kps_left,
    kps_right=kps_right, joints_left=joints_left, joints_right=joints_right)
    print('INFO: Testing on {} frames'.format(test_generator.num_frames()))
4
6
    if not args.evaluate:
        cameras_train, poses_train, poses_train_2d = fetch(subjects_train,
    action filter, subset=args.subset)
 8
9
        lr = args.learning_rate
10
        if semi supervised:
```

```
11
            cameras semi, , poses semi 2d = fetch(subjects semi,
    action filter, parse 3d poses=False)
12
13
            if not args.disable optimizations and not args.dense and
    args.stride == 1:
                # Use optimized model for single-frame predictions
14
15
                model_traj_train =
    TemporalModelOptimized1f(poses valid 2d[0].shape[-2],
    poses_valid_2d[0].shape[-1], 1,
16
                         filter_widths=filter_widths, causal=args.causal,
    dropout=args.dropout, channels=args.channels)
17
            else:
                # When incompatible settings are detected (stride > 1, dense
18
    filters, or disabled optimization) fall back to normal model
19
                model traj train = TemporalModel(poses valid 2d[0].shape[-2],
    poses valid 2d[0].shape[-1], 1,
20
                         filter_widths=filter_widths, causal=args.causal,
    dropout=args.dropout, channels=args.channels,
21
                        dense=args.dense)
22
            model traj = TemporalModel(poses valid 2d[0].shape[-2],
23
    poses valid 2d[0].shape[-1], 1,
                                 filter_widths=filter_widths,
24
    causal=args.causal, dropout=args.dropout, channels=args.channels,
25
                                 dense=args.dense)
            if torch.cuda.is available():
26
27
                model_traj = model_traj.cuda()
28
                model traj train = model traj train.cuda()
29
            optimizer = optim.Adam(list(model pos train.parameters()) +
    list(model traj train.parameters()),
30
                                    lr=lr, amsgrad=True)
31
32
            losses_2d_train_unlabeled = []
33
            losses 2d train labeled eval = []
            losses 2d train unlabeled eval = []
34
35
            losses_2d_valid = []
36
37
            losses_traj_train = []
            losses traj train eval = []
38
39
            losses_traj_valid = []
40
        else:
41
            optimizer = optim.Adam(model pos train.parameters(), lr=lr,
    amsgrad=True)
42
43
        lr decay = args.lr decay
44
45
        losses_3d_train = []
46
        losses_3d_train_eval = []
47
        losses 3d valid = []
```

```
48
49
        epoch = 0
50
        initial momentum = 0.1
51
        final momentum = 0.001
52
53
54
        train_generator = ChunkedGenerator(args.batch_size//args.stride,
    cameras train, poses train, poses train 2d, args.stride,
55
                                            pad=pad,
    causal_shift=causal_shift, shuffle=True, augment=args.data_augmentation,
56
                                            kps left=kps left,
    kps_right=kps_right, joints_left=joints_left, joints_right=joints_right)
        train_generator_eval = UnchunkedGenerator(cameras_train, poses_train,
57
    poses_train_2d,
58
                                                   pad=pad,
    causal shift=causal shift, augment=False)
59
        print('INFO: Training on {}
    frames'.format(train generator eval.num frames()))
60
        if semi supervised:
61
            semi_generator = ChunkedGenerator(args.batch_size//args.stride,
    cameras semi, None, poses semi 2d, args.stride,
62
                                               pad=pad,
    causal_shift=causal_shift, shuffle=True,
63
                                               random_seed=4321,
    augment=args.data augmentation,
64
                                               kps left=kps left,
    kps_right=kps_right, joints_left=joints_left, joints_right=joints_right,
65
                                               endless=True)
66
            semi generator eval = UnchunkedGenerator(cameras semi, None,
    poses_semi_2d,
67
                                                      pad=pad,
    causal shift=causal shift, augment=False)
68
            print('INFO: Semi-supervision on {}
    frames'.format(semi_generator_eval.num_frames()))
69
7.0
        if args.resume:
71
            epoch = checkpoint['epoch']
72
            if 'optimizer' in checkpoint and checkpoint['optimizer'] is not
    None:
73
                optimizer.load_state_dict(checkpoint['optimizer'])
74
                train_generator.set_random_state(checkpoint['random_state'])
75
            else:
76
                print('WARNING: this checkpoint does not contain an optimizer
    state. The optimizer will be reinitialized.')
77
            lr = checkpoint['lr']
78
79
            if semi supervised:
80
                model_traj_train.load_state_dict(checkpoint['model_traj'])
81
                model traj.load state dict(checkpoint['model traj'])
```

```
82
      semi generator.set random state(checkpoint['random state semi'])
 83
 84
         print('** Note: reported losses are averaged over all frames and
     test-time augmentation is not used here.')
         print('** The final evaluation will be carried out after the last
 85
     training epoch.')
 86
         # Pos model only
 87
 88
         while epoch < args.epochs:
 89
             start time = time()
 90
             epoch_loss_3d_train = 0
             epoch_loss_traj_train = 0
 91
             epoch_loss_2d_train_unlabeled = 0
 92
 93
             N = 0
             N_semi = 0
 94
 95
             model_pos_train.train()
 96
             if semi supervised:
 97
                 # Semi-supervised scenario
                 model_traj_train.train()
 98
 99
                 for (_, batch_3d, batch_2d), (cam_semi, _, batch_2d_semi) in
100
                      zip(train_generator.next_epoch(),
     semi_generator.next_epoch()):
101
                      # Fall back to supervised training for the first epoch
102
     (to avoid instability)
103
                      skip = epoch < args.warmup</pre>
104
105
                      cam_semi = torch.from_numpy(cam_semi.astype('float32'))
106
                      inputs_3d = torch.from_numpy(batch_3d.astype('float32'))
107
                      if torch.cuda.is available():
108
                          cam_semi = cam_semi.cuda()
109
                          inputs_3d = inputs_3d.cuda()
110
111
                      inputs_traj = inputs_3d[:, :, :1].clone()
112
                      inputs 3d[:, :, 0] = 0
113
                      # Split point between labeled and unlabeled samples in
114
     the batch
115
                      split_idx = inputs_3d.shape[0]
116
117
                      inputs_2d = torch.from_numpy(batch_2d.astype('float32'))
118
                      inputs 2d semi =
     torch.from_numpy(batch_2d_semi.astype('float32'))
                      if torch.cuda.is_available():
119
                          inputs 2d = inputs 2d.cuda()
120
121
                          inputs_2d_semi = inputs_2d_semi.cuda()
```

```
122
                      inputs 2d cat = torch.cat((inputs 2d, inputs 2d semi),
     dim=0) if not skip else inputs 2d
123
124
                      optimizer.zero grad()
125
126
                      # Compute 3D poses
127
                      predicted_3d_pos_cat = model_pos_train(inputs_2d_cat)
128
129
                      loss_3d_pos = mpjpe(predicted_3d_pos_cat[:split_idx],
     inputs_3d)
130
                      epoch loss 3d train +=
     inputs_3d.shape[0]*inputs_3d.shape[1] * loss_3d_pos.item()
                      N += inputs_3d.shape[0]*inputs_3d.shape[1]
131
132
                      loss_total = loss_3d_pos
133
134
                      # Compute global trajectory
135
                      predicted_traj_cat = model_traj_train(inputs_2d_cat)
136
                     w = 1 / inputs traj[:, :, :, 2] # Weight inversely
     proportional to depth
                      loss_traj =
137
     weighted mpjpe(predicted traj cat[:split idx], inputs traj, w)
138
                     epoch loss traj train +=
     inputs_3d.shape[0]*inputs_3d.shape[1] * loss_traj.item()
139
                      assert inputs_traj.shape[0]*inputs_traj.shape[1] ==
     inputs 3d.shape[0]*inputs 3d.shape[1]
                     loss total += loss traj
140
141
142
                      if not skip:
143
                          # Semi-supervised loss for unlabeled samples
144
                         predicted_semi = predicted_3d_pos_cat[split_idx:]
145
                          if pad > 0:
146
                              target semi = inputs 2d semi[:, pad:-pad, :,
     :2].contiguous()
147
                         else:
148
                              target semi = inputs 2d semi[:, :, :,
     :2].contiguous()
149
150
                         projection_func = project_to_2d_linear if
     args.linear projection else project to 2d
151
                          reconstruction_semi = projection_func(predicted_semi
     + predicted_traj_cat[split_idx:], cam_semi)
152
                          loss_reconstruction = mpjpe(reconstruction_semi,
153
     target semi) # On 2D poses
154
                          epoch_loss_2d_train_unlabeled +=
     predicted_semi.shape[0]*predicted_semi.shape[1] *
     loss reconstruction.item()
                          if not args.no_proj:
155
156
                              loss total += loss reconstruction
```

```
157
158
                          # Bone length term to enforce kinematic constraints
                          if args.bone length term:
159
160
                              dists = predicted 3d pos cat[:, :, 1:] -
     predicted_3d_pos_cat[:, :, dataset.skeleton().parents()[1:]]
                              bone lengths = torch.mean(torch.norm(dists,
161
     dim=3), dim=1)
162
                              penalty =
     torch.mean(torch.abs(torch.mean(bone_lengths[:split_idx], dim=0) \
163
     torch.mean(bone lengths[split idx:], dim=0)))
164
                              loss total += penalty
165
166
167
                          N semi +=
     predicted_semi.shape[0]*predicted_semi.shape[1]
168
                      else:
169
                          N semi += 1 # To avoid division by zero
170
171
                      loss_total.backward()
172
                      optimizer.step()
173
174
                 losses_traj_train.append(epoch_loss_traj_train / N)
175
      losses 2d train unlabeled.append(epoch loss 2d train unlabeled / N semi)
             else:
176
                 # Regular supervised scenario
177
178
                 for , batch 3d, batch 2d in train generator.next epoch():
                      inputs 3d = torch.from numpy(batch 3d.astype('float32'))
179
180
                      inputs_2d = torch.from_numpy(batch_2d.astype('float32'))
181
                      if torch.cuda.is_available():
182
                          inputs 3d = inputs 3d.cuda()
183
                          inputs_2d = inputs_2d.cuda()
                      inputs_3d[:, :, 0] = 0
184
185
186
                     optimizer.zero_grad()
187
                      # Predict 3D poses
188
                      predicted 3d pos = model pos train(inputs 2d)
189
190
                      loss_3d_pos = mpjpe(predicted_3d_pos, inputs_3d)
191
                      epoch_loss_3d_train +=
     inputs 3d.shape[0]*inputs 3d.shape[1] * loss 3d pos.item()
192
                      N += inputs_3d.shape[0]*inputs_3d.shape[1]
193
194
                      loss total = loss 3d pos
195
                      loss_total.backward()
196
197
                      optimizer.step()
198
```

```
199
              losses 3d train.append(epoch loss 3d train / N)
200
             # End-of-epoch evaluation
201
             with torch.no grad():
2.02
2.03
                  model_pos.load_state_dict(model_pos_train.state_dict())
204
                  model pos.eval()
205
                  if semi_supervised:
                      model_traj.load_state_dict(model_traj_train.state_dict())
206
207
                      model traj.eval()
208
209
                  epoch loss 3d valid = 0
210
                  epoch loss traj valid = 0
                  epoch_loss_2d_valid = 0
211
                  N = 0
212
2.13
214
                  if not args.no eval:
215
                      # Evaluate on test set
216
                      for cam, batch, batch 2d in test generator.next epoch():
217
                          inputs 3d = torch.from numpy(batch.astype('float32'))
                          inputs_2d =
218
     torch.from numpy(batch 2d.astype('float32'))
219
                          if torch.cuda.is available():
220
                              inputs 3d = inputs 3d.cuda()
221
                              inputs_2d = inputs_2d.cuda()
2.2.2
                          inputs traj = inputs 3d[:, :, :1].clone()
223
                          inputs 3d[:, :, 0] = 0
224
225
                          # Predict 3D poses
                          predicted 3d pos = model pos(inputs 2d)
2.2.6
227
                          loss_3d_pos = mpjpe(predicted_3d_pos, inputs_3d)
228
                          epoch_loss_3d_valid +=
     inputs 3d.shape[0]*inputs 3d.shape[1] * loss 3d pos.item()
229
                          N += inputs_3d.shape[0]*inputs_3d.shape[1]
230
231
                          if semi supervised:
2.32
                              cam = torch.from_numpy(cam.astype('float32'))
233
                              if torch.cuda.is available():
234
                                  cam = cam.cuda()
235
236
                              predicted_traj = model_traj(inputs_2d)
237
                              loss_traj = mpjpe(predicted_traj, inputs_traj)
238
                              epoch loss traj valid +=
     inputs_traj.shape[0]*inputs_traj.shape[1] * loss_traj.item()
                              assert inputs traj.shape[0]*inputs traj.shape[1]
239
     == inputs 3d.shape[0]*inputs 3d.shape[1]
2.40
                              if pad > 0:
241
242
                                   target = inputs_2d[:, pad:-pad, :,
     :2].contiguous()
```

```
243
                              else:
244
                                  target = inputs 2d[:, :, :, :2].contiguous()
245
                              reconstruction = project to 2d(predicted 3d pos +
     predicted traj, cam)
246
                              loss_reconstruction = mpjpe(reconstruction,
     target) # On 2D poses
247
                              epoch_loss_2d_valid +=
     reconstruction.shape[0]*reconstruction.shape[1] *
     loss reconstruction.item()
248
                              assert
     reconstruction.shape[0]*reconstruction.shape[1] ==
     inputs 3d.shape[0]*inputs 3d.shape[1]
249
                      losses_3d_valid.append(epoch_loss_3d_valid / N)
250
2.51
                      if semi supervised:
252
                          losses traj valid.append(epoch loss traj valid / N)
253
                          losses_2d_valid.append(epoch_loss_2d_valid / N)
254
255
256
                      # Evaluate on training set, this time in evaluation mode
                      epoch_loss_3d_train_eval = 0
257
                      epoch loss traj train eval = 0
258
259
                      epoch_loss_2d_train_labeled_eval = 0
260
                      N = 0
2.61
                      for cam, batch, batch 2d in
     train_generator_eval.next_epoch():
262
                          if batch 2d.shape[1] == 0:
2.63
                              # This can only happen when downsampling the
     dataset
264
                              continue
265
2.66
                          inputs 3d = torch.from numpy(batch.astype('float32'))
267
                          inputs_2d =
     torch.from_numpy(batch_2d.astype('float32'))
                          if torch.cuda.is available():
2.68
269
                              inputs_3d = inputs_3d.cuda()
                              inputs 2d = inputs 2d.cuda()
270
271
                          inputs_traj = inputs_3d[:, :, :1].clone()
                          inputs 3d[:, :, 0] = 0
272
273
                          # Compute 3D poses
274
275
                          predicted 3d pos = model pos(inputs 2d)
276
                          loss_3d_pos = mpjpe(predicted_3d_pos, inputs_3d)
                          epoch loss 3d train eval +=
277
     inputs_3d.shape[0]*inputs_3d.shape[1] * loss_3d_pos.item()
                          N += inputs_3d.shape[0]*inputs_3d.shape[1]
278
279
280
                          if semi_supervised:
281
                              cam = torch.from numpy(cam.astype('float32'))
```

```
282
                              if torch.cuda.is available():
283
                                  cam = cam.cuda()
                              predicted traj = model traj(inputs 2d)
284
                              loss traj = mpjpe(predicted traj, inputs traj)
2.85
2.86
                              epoch_loss_traj_train_eval +=
     inputs traj.shape[0]*inputs traj.shape[1] * loss traj.item()
287
                              assert inputs_traj.shape[0]*inputs_traj.shape[1]
     == inputs 3d.shape[0]*inputs 3d.shape[1]
288
289
                              if pad > 0:
290
                                  target = inputs 2d[:, pad:-pad, :,
     :2].contiguous()
291
                              else:
292
                                  target = inputs_2d[:, :, :, :2].contiguous()
293
                              reconstruction = project_to_2d(predicted_3d_pos +
     predicted_traj, cam)
294
                              loss_reconstruction = mpjpe(reconstruction,
     target)
295
                              epoch loss 2d train labeled eval +=
     reconstruction.shape[0]*reconstruction.shape[1] *
     loss reconstruction.item()
296
                              assert
     reconstruction.shape[0]*reconstruction.shape[1] ==
     inputs_3d.shape[0]*inputs_3d.shape[1]
297
298
                      losses_3d_train_eval.append(epoch_loss_3d_train_eval / N)
                      if semi_supervised:
299
300
      losses traj train eval.append(epoch loss traj train eval / N)
301
      losses_2d_train_labeled_eval.append(epoch_loss_2d_train_labeled_eval /
     N)
302
303
                      # Evaluate 2D loss on unlabeled training set (in
     evaluation mode)
304
                      epoch_loss_2d_train_unlabeled_eval = 0
305
                      N \text{ semi} = 0
306
                      if semi_supervised:
                          for cam, , batch 2d in
307
     semi_generator_eval.next_epoch():
308
                              cam = torch.from_numpy(cam.astype('float32'))
309
                              inputs 2d semi =
     torch.from_numpy(batch_2d.astype('float32'))
310
                              if torch.cuda.is available():
311
                                  cam = cam.cuda()
                                  inputs_2d_semi = inputs_2d_semi.cuda()
312
313
314
                              predicted_3d_pos_semi = model_pos(inputs_2d_semi)
315
                              predicted traj semi = model traj(inputs 2d semi)
```

```
316
                              if pad > 0:
317
                                  target semi = inputs 2d semi[:, pad:-pad, :,
     :2].contiguous()
318
                              else:
319
                                  target_semi = inputs_2d_semi[:, :, :,
     :2].contiguous()
320
                              reconstruction_semi =
     project to 2d(predicted 3d pos semi + predicted traj semi, cam)
                              loss reconstruction semi =
321
     mpjpe(reconstruction_semi, target_semi)
322
323
                              epoch loss 2d train unlabeled eval +=
     reconstruction_semi.shape[0]*reconstruction_semi.shape[1] \
324
     loss reconstruction semi.item()
325
                              N semi +=
     reconstruction_semi.shape[0]*reconstruction_semi.shape[1]
326
      losses 2d train unlabeled eval.append(epoch loss 2d train unlabeled eval
     / N_semi)
327
             elapsed = (time() - start time)/60
328
329
             if args.no_eval:
330
                 print('[%d] time %.2f lr %f 3d train %f' % (
331
332
                          epoch + 1,
333
                          elapsed,
334
                          lr,
335
                          losses 3d train[-1] * 1000))
336
             else:
337
                 if semi_supervised:
338
                      print('[%d] time %.2f lr %f 3d train %f 3d eval %f
     traj eval %f 3d valid %f '
339
                            'traj valid %f 2d train sup %f 2d train unsup %f
     2d valid %f' % (
340
                              epoch + 1,
341
                              elapsed,
342
                              lr,
343
                              losses 3d train[-1] * 1000,
                              losses 3d_train_eval[-1] * 1000,
344
345
                              losses_traj_train_eval[-1] * 1000,
346
                              losses 3d valid[-1] * 1000,
                              losses_traj_valid[-1] * 1000,
347
                              losses 2d train labeled eval[-1],
348
349
                              losses_2d_train_unlabeled_eval[-1],
350
                              losses_2d_valid[-1]))
351
                 else:
352
                      print('[%d] time %.2f lr %f 3d_train %f 3d_eval %f
     3d valid %f' % (
```

```
353
                              epoch + 1,
354
                              elapsed,
                              lr,
355
356
                              losses 3d train[-1] * 1000,
357
                              losses_3d_train_eval[-1] * 1000,
                              losses 3d valid[-1] *1000))
358
359
             # Decay learning rate exponentially
360
             lr *= lr decay
361
362
             for param_group in optimizer.param_groups:
363
                 param group['lr'] *= lr decay
364
             epoch += 1
365
366
             # Decay BatchNorm momentum
367
             momentum = initial momentum * np.exp(-epoch/args.epochs *
     np.log(initial momentum/final momentum))
368
             model_pos_train.set_bn_momentum(momentum)
369
             if semi supervised:
370
                 model traj train.set bn momentum(momentum)
371
             # Save checkpoint if necessary
372
             if epoch % args.checkpoint frequency == 0:
373
                 chk_path = os.path.join(args.checkpoint,
374
     'epoch_{}.bin'.format(epoch))
                 print('Saving checkpoint to', chk path)
375
376
377
                 torch.save({
378
                      'epoch': epoch,
                      'lr': lr,
379
380
                      'random state': train generator.random state(),
381
                      'optimizer': optimizer.state_dict(),
382
                      'model pos': model pos train.state dict(),
383
                      'model_traj': model_traj_train.state_dict() if
     semi supervised else None,
                      'random state semi': semi generator.random state() if
384
     semi_supervised else None,
385
                 }, chk path)
386
             # Save training curves after every epoch, as .png images (if
387
     requested)
             if args.export_training_curves and epoch > 3:
388
                 if 'matplotlib' not in sys.modules:
389
390
                      import matplotlib
391
                      matplotlib.use('Agg')
392
                      import matplotlib.pyplot as plt
393
394
                 plt.figure()
                 epoch_x = np.arange(3, len(losses_3d_train)) + 1
395
396
                 plt.plot(epoch x, losses 3d train[3:], '--', color='C0')
```

```
397
                 plt.plot(epoch x, losses 3d train eval[3:], color='C0')
398
                 plt.plot(epoch x, losses 3d valid[3:], color='C1')
                 plt.legend(['3d train', '3d train (eval)', '3d valid
399
     (eval)'])
400
                 plt.ylabel('MPJPE (m)')
401
                 plt.xlabel('Epoch')
402
                 plt.xlim((3, epoch))
                 plt.savefig(os.path.join(args.checkpoint, 'loss 3d.png'))
403
404
                 if semi_supervised:
405
406
                      plt.figure()
407
                      plt.plot(epoch_x, losses_traj_train[3:], '--',
     color='C0')
                      plt.plot(epoch_x, losses_traj_train_eval[3:], color='C0')
408
                      plt.plot(epoch_x, losses_traj_valid[3:], color='C1')
409
                      plt.legend(['traj. train', 'traj. train (eval)', 'traj.
410
     valid (eval)'])
411
                      plt.ylabel('Mean distance (m)')
412
                      plt.xlabel('Epoch')
                      plt.xlim((3, epoch))
413
414
                      plt.savefig(os.path.join(args.checkpoint,
     'loss traj.png'))
415
416
                      plt.figure()
417
                      plt.plot(epoch x, losses 2d train labeled eval[3:],
     color='C0')
418
                      plt.plot(epoch_x, losses_2d_train_unlabeled[3:], '--',
     color='C1')
419
                      plt.plot(epoch x, losses 2d train unlabeled eval[3:],
     color='C1')
420
                      plt.plot(epoch_x, losses_2d_valid[3:], color='C2')
421
                      plt.legend(['2d train labeled (eval)', '2d train
     unlabeled', '2d train unlabeled (eval)', '2d valid (eval)'])
422
                      plt.ylabel('MPJPE (2D)')
                      plt.xlabel('Epoch')
423
424
                      plt.xlim((3, epoch))
425
                      plt.savefig(os.path.join(args.checkpoint, 'loss_2d.png'))
426
                 plt.close('all')
```

Evaluate(本部分不要求理解,只要会用就行)

```
# Evaluate
    def evaluate(test_generator, action=None, return_predictions=False,
    use_trajectory_model=False):
        epoch loss 3d pos = 0
 3
4
        epoch_loss_3d_pos_procrustes = 0
 5
        epoch_loss_3d_pos_scale = 0
        epoch loss 3d vel = 0
 6
7
        with torch.no_grad():
            if not use trajectory model:
8
                model pos.eval()
10
            else:
11
                model traj.eval()
12
            N = 0
            for , batch, batch 2d in test generator.next epoch():
13
                inputs 2d = torch.from numpy(batch 2d.astype('float32'))
14
                if torch.cuda.is_available():
15
                     inputs_2d = inputs_2d.cuda()
17
18
                # Positional model
19
                if not use_trajectory_model:
20
                     predicted_3d_pos = model_pos(inputs_2d)
21
                else:
                    predicted_3d_pos = model_traj(inputs_2d)
22
23
24
                # Test-time augmentation (if enabled)
                if test_generator.augment_enabled():
25
26
                    # Undo flipping and take average with non-flipped version
                    predicted_3d_pos[1, :, :, 0] *= -1
2.7
28
                    if not use trajectory model:
29
                         predicted_3d_pos[1, :, joints_left + joints_right] =
    predicted_3d_pos[1, :, joints_right + joints_left]
30
                     predicted 3d pos = torch.mean(predicted 3d pos, dim=0,
    keepdim=True)
31
                if return predictions:
32
33
                    return predicted_3d_pos.squeeze(0).cpu().numpy()
34
                inputs 3d = torch.from numpy(batch.astype('float32'))
35
36
                if torch.cuda.is available():
                     inputs_3d = inputs_3d.cuda()
37
38
                inputs 3d[:, :, 0] = 0
39
                if test generator.augment enabled():
```

```
40
                    inputs 3d = inputs 3d[:1]
41
                error = mpjpe(predicted_3d_pos, inputs_3d)
42
43
                epoch_loss_3d_pos_scale +=
    inputs_3d.shape[0]*inputs_3d.shape[1] * n_mpjpe(predicted_3d_pos,
    inputs 3d).item()
44
                epoch loss 3d pos += inputs 3d.shape[0]*inputs 3d.shape[1] *
45
    error.item()
46
                N += inputs_3d.shape[0] * inputs_3d.shape[1]
47
                inputs = inputs_3d.cpu().numpy().reshape(-1,
48
    inputs_3d.shape[-2], inputs_3d.shape[-1])
49
                predicted_3d_pos = predicted_3d_pos.cpu().numpy().reshape(-1,
    inputs_3d.shape[-2], inputs_3d.shape[-1])
50
51
                epoch_loss_3d_pos_procrustes +=
    inputs_3d.shape[0]*inputs_3d.shape[1] * p_mpjpe(predicted_3d_pos, inputs)
52
53
                # Compute velocity error
54
                epoch loss 3d vel += inputs 3d.shape[0]*inputs 3d.shape[1] *
    mean_velocity_error(predicted_3d_pos, inputs)
55
56
        if action is None:
57
            print('----')
        else:
58
            print('----'+action+'----')
59
60
        e1 = (epoch_loss_3d_pos / N)*1000
        e2 = (epoch loss 3d pos procrustes / N)*1000
61
62
        e3 = (epoch_loss_3d_pos_scale / N)*1000
        ev = (epoch_loss_3d_vel / N)*1000
63
64
        print('Test time augmentation:', test generator.augment enabled())
65
        print('Protocol #1 Error (MPJPE):', e1, 'mm')
        print('Protocol #2 Error (P-MPJPE):', e2, 'mm')
66
        print('Protocol #3 Error (N-MPJPE):', e3, 'mm')
67
        print('Velocity Error (MPJVE):', ev, 'mm')
68
69
        print('----')
70
71
        return e1, e2, e3, ev
72
73
74
    if args.render:
75
        print('Rendering...')
76
77
        input_keypoints = keypoints[args.viz_subject][args.viz_action]
    [args.viz_camera].copy()
78
        ground_truth = None
79
        if args.viz_subject in dataset.subjects() and args.viz_action in
    dataset[args.viz_subject]:
```

```
80
             if 'positions 3d' in dataset[args.viz subject][args.viz action]:
 81
                 ground truth = dataset[args.viz subject][args.viz action]
     ['positions 3d'][args.viz camera].copy()
         if ground truth is None:
 82
 83
             print('INFO: this action is unlabeled. Ground truth will not be
     rendered.')
 84
         gen = UnchunkedGenerator(None, None, [input keypoints],
 85
 86
                                   pad=pad, causal shift=causal shift,
     augment=args.test_time_augmentation,
 87
                                   kps left=kps left, kps right=kps right,
     joints left=joints left, joints right=joints right)
         prediction = evaluate(gen, return_predictions=True)
 88
         if model traj is not None and ground truth is None:
 89
             prediction_traj = evaluate(gen, return_predictions=True,
 90
     use trajectory model=True)
 91
             prediction += prediction_traj
 92
 93
         if args.viz export is not None:
             print('Exporting joint positions to', args.viz_export)
 94
 95
             # Predictions are in camera space
             np.save(args.viz export, prediction)
 96
 97
         if args.viz_output is not None:
 98
 99
             if ground truth is not None:
100
                 # Reapply trajectory
                 trajectory = ground_truth[:, :1]
101
102
                 ground truth[:, 1:] += trajectory
103
                 prediction += trajectory
104
105
             # Invert camera transformation
106
             cam = dataset.cameras()[args.viz subject][args.viz camera]
107
             if ground_truth is not None:
108
                 prediction = camera to world(prediction,
     R=cam['orientation'], t=cam['translation'])
109
                 ground_truth = camera_to_world(ground_truth,
     R=cam['orientation'], t=cam['translation'])
110
             else:
                 # If the ground truth is not available, take the camera
111
     extrinsic params from a random subject.
112
                 # They are almost the same, and anyway, we only need this for
     visualization purposes.
                 for subject in dataset.cameras():
113
114
                      if 'orientation' in dataset.cameras()[subject]
     [args.viz_camera]:
                         rot = dataset.cameras()[subject][args.viz camera]
115
     ['orientation']
116
                          break
117
                 prediction = camera to world(prediction, R=rot, t=0)
```

```
118
                 # We don't have the trajectory, but at least we can rebase
     the height
119
                 prediction[:, :, 2] -= np.min(prediction[:, :, 2])
120
121
             anim output = {'Reconstruction': prediction}
122
             if ground truth is not None and not args.viz no ground truth:
123
                 anim_output['Ground truth'] = ground_truth
124
125
             input keypoints = image coordinates(input keypoints[..., :2],
     w=cam['res_w'], h=cam['res_h'])
126
             from common.visualization import render animation
127
             render_animation(input_keypoints, keypoints_metadata,
128
     anim output,
129
                               dataset.skeleton(), dataset.fps(),
     args.viz bitrate, cam['azimuth'], args.viz output,
130
                               limit=args.viz_limit,
     downsample=args.viz downsample, size=args.viz size,
131
                               input video path=args.viz video, viewport=
     (cam['res_w'], cam['res_h']),
132
                               input video skip=args.viz skip)
133
     else:
134
135
         print('Evaluating...')
136
         all actions = {}
         all actions by subject = {}
137
         for subject in subjects_test:
138
139
             if subject not in all actions by subject:
140
                 all actions by subject[subject] = {}
141
             for action in dataset[subject].keys():
142
143
                 action name = action.split(' ')[0]
144
                 if action_name not in all_actions:
145
                      all actions[action name] = []
                 if action name not in all actions by subject[subject]:
146
147
                      all_actions_by_subject[subject][action_name] = []
148
                 all actions[action name].append((subject, action))
149
                 all_actions_by_subject[subject][action_name].append((subject,
     action))
150
151
         def fetch_actions(actions):
             out poses 3d = []
152
             out_poses_2d = []
153
154
             for subject, action in actions:
155
156
                 poses_2d = keypoints[subject][action]
                 for i in range(len(poses 2d)): # Iterate across cameras
157
158
                      out_poses_2d.append(poses_2d[i])
159
```

```
160
                 poses 3d = dataset[subject][action]['positions 3d']
161
                 assert len(poses 3d) == len(poses 2d), 'Camera count
     mismatch'
                 for i in range(len(poses 3d)): # Iterate across cameras
162
163
                      out_poses_3d.append(poses_3d[i])
164
165
             stride = args.downsample
             if stride > 1:
166
                 # Downsample as requested
167
168
                 for i in range(len(out_poses_2d)):
169
                      out poses 2d[i] = out poses 2d[i][::stride]
170
                      if out poses 3d is not None:
171
                          out_poses_3d[i] = out_poses_3d[i][::stride]
172
173
             return out_poses_3d, out_poses_2d
174
175
         def run_evaluation(actions, action_filter=None):
176
             errors p1 = []
             errors p2 = []
177
178
             errors_p3 = []
179
             errors vel = []
180
             for action_key in actions.keys():
181
                 if action_filter is not None:
182
183
                      found = False
                      for a in action filter:
184
                          if action_key.startswith(a):
185
186
                              found = True
187
                              break
188
                      if not found:
189
                          continue
190
191
                 poses_act, poses_2d_act = fetch_actions(actions[action_key])
192
                 gen = UnchunkedGenerator(None, poses_act, poses_2d_act,
193
                                           pad=pad, causal shift=causal shift,
     augment=args.test_time_augmentation,
194
                                            kps left=kps left,
     kps_right=kps_right, joints_left=joints_left, joints_right=joints_right)
195
                 e1, e2, e3, ev = evaluate(gen, action key)
196
                 errors_pl.append(e1)
197
                 errors_p2.append(e2)
198
                 errors p3.append(e3)
199
                 errors_vel.append(ev)
200
201
             print('Protocol #1 (MPJPE) action-wise average:',
     round(np.mean(errors_p1), 1), 'mm')
             print('Protocol #2 (P-MPJPE) action-wise average:',
202
     round(np.mean(errors_p2), 1), 'mm')
```

```
print('Protocol #3 (N-MPJPE) action-wise average:',
203
     round(np.mean(errors_p3), 1), 'mm')
204
             print('Velocity (MPJVE) action-wise average:',
     round(np.mean(errors_vel), 2), 'mm')
205
         if not args.by_subject:
206
207
             run_evaluation(all_actions, action_filter)
         else:
208
             for subject in all_actions_by_subject.keys():
209
210
                 print('Evaluating on subject', subject)
                 run_evaluation(all_actions_by_subject[subject],
211
     action_filter)
212
                 print('')
```